

Incorporating Integrity Constraints in Uncertain Databases

Naveen Ashish¹, Sharad Mehrotra², Pouria Pirzadeh³

Calit2 and ICS Department, UC Irvine
Irvine CA 92697 USA

¹ashish@ics.uci.edu

²sharad@ics.uci.edu

Abstract— We develop an approach to incorporate additional knowledge, in the form of general-purpose integrity constraints (ICs), to reduce uncertainty in probabilistic databases. While incorporating ICs improves data quality (and hence quality of answers to a query), it significantly complicates query processing. To overcome the additional complexity, we develop an approach to map an uncertain relation U with ICs to another uncertain relation U' that approximates the set of consistent worlds represented by U . Queries over U can instead be evaluated over U' achieving higher quality (due to reduced uncertainty in U') without additional complexity in query processing due to ICs. We demonstrate the effectiveness and scalability of our approach to large datasets with complex constraints. We also present experimental results demonstrating the utility of incorporating integrity constraints in uncertain relations, in the context of an information extraction application.

I. INTRODUCTION

Recent advances in probabilistic models for information extraction, document classification, and automated tagging has revived significant interest in probabilistic data management. Extraction techniques based on models such as conditional random fields (CRFs) [1] create a database wherein tuples and/or attribute values have associated explicit estimates of probability. Multiple probabilistic models [2]–[4], of varying expressivity, have been developed to represent such uncertain data along with efficient query processing approaches [2], [3], [5] to support search and analysis capability on such uncertain databases. In this paper we develop an approach to incorporating additional semantics, in the form of database *integrity constraints*, that can reduce uncertainty in data, which, in turn, could positively impact applications such as query answering and retrieval

Consider the example in Fig 1 where we have an Employee relation with uncertainty, represented using “or-sets” [2] for each attribute. For instance in tuple 1 the job-title of the employee “jim” is *either* instructor (with probability 0.7) or a manager (with probability 0.3). This uncertain relation represents 4 *possible worlds* which are the 4 possibilities of this relation based on different attribute value choices in each attribute. A query Q over such a probabilistic database returns tuples that satisfy Q in one of the possible worlds along with their corresponding probabilities. Now consider additional semantics in the form of say a functional dependency (FD) that states that a person cannot

Employee			
name	job-title	division	degree
jim	instructor(0.7) manager(0.3)	training	MBA
jim	manager	marketing	MBA
jim (0.5) jill (0.5)	consultant	innovation	PhD

Constraint (C): (name, job-title) \rightarrow division

Fig. 1. Uncertain Relation with Constraints

hold the same job title in 2 different divisions, i.e., (name, job-title) \rightarrow division. Given this knowledge, we know that two out of the four possible worlds, where the first tuple has “Jim” as a “manager” (in “training”), are impossible as they violate the functional dependency. A natural extension to the query semantics is to return *only* those tuples that satisfy the query Q in one of the *consistent* possible worlds. As a result, the answer to the query about “jim” above should not include the tuple $\{jim, manager, training\}$ since such a tuple is not part of any consistent instance of the relation.

Incorporating additional knowledge such as constraints to reduce uncertainty in query results has indeed been explored to various degrees in different uncertain database models and systems [3], [6], [7]. For instance Trio [3] and [6] permit the specification of constraints, but at the data *instance* level i.e., between individual attribute or tuple instances. For instance using the notation T1(2) to represent the second tuple instance (possibility) in the first tuple in Fig 1 i.e., (jim, manager, training, MBA) and T2(1) to represent the first (and only) tuple instance in the second tuple, we could specify a constraint such as T1(2) XOR T2(1) which states that only one of these tuple instances can exist together in a possible world. Query answering approaches for such models [6] can address only a small number (< 20) of such constraint instances. [8] considers very restricted forms of FD and IND constraints in addition to database statistics, to address a different problem - that of determining a maximum likelihood *estimate* of the probability of a query answer in a data integration setting. MayBMS [7] has considered more general integrity constraints at the level of individual tuples as well as functional dependencies in their probabilistic model

based on representing uncertain databases using world set decompositions. Their approach for factoring FDs however can be shown to be exponential - as we illustrate in the related work section. This is not surprising as it can be shown that answering even simple selection queries exactly over uncertain relation in presence of integrity constraints (e.g., a single functional dependency), is NP-hard. We state the following:

Statement 1: Given a U-relation, U , and a functional dependency (FD) F defined over U , identifying a possible world instance $pw_q \in PW_U$ such that $pw_q \models F$ or determining that no such instance exists is NP-Hard. We refer to this problem as the **The FD consistency problem**.

Proof: This follows by a reduction from the 3-SAT problem which is known to be NP-Hard. The proof is as follows:

1. Given an instance of 3-SAT i.e., a CNF expression:
 $(x_{11} \vee x_{12} \vee x_{13}) \wedge (x_{21} \vee x_{22} \vee x_{23}) \dots \wedge (x_{n1} \vee x_{n2} \vee x_{n3})$.
 We will now create a corresponding uncertain relation U as follows.

2. Consider any one conjunct, each conjunct is of the form of one of $(x_1 \vee x_2 \vee x_3)$, $(x_{11} \vee x_2 \vee \neg x_3)$, $(\neg x_{11} \vee \neg x_2 \vee x_3)$, or $(\neg x_{11} \vee \neg x_2 \vee \neg x_3)$

3. Take the following actions based on the type of the conjunct:
 (i) Type is: $(x_1 \vee x_2 \vee x_3)$

Create the following 3 uncertain tuples, each tuple with 3 tuple instances (choices), in U :

$\{T_{x1}, T_{x2}, T_{x3}\}$
 $\{T_{x1}, T_{x2}, T_{x3}\}$
 $\{T_{x1}, T_{x2}, T_{x3}\}$

Let the tuple have some attributes (which are at least 3 in number). Let the first attribute be a tuple instance identifier (ID), tuple instance T_{x1} has ID 1, tuple instance T_{x2} has ID 2, etc.

(ii) Type is: $(x_1 \vee x_2 \vee \neg x_3)$

Note that we can also treat this as $x_3 \rightarrow (x_1 \vee x_2)$

Create the following tuple instances in U :

$\{T_{x1}, T_{x2}, \neg T_{x3}\}$
 $\{T_{x1}, T_{x2}, \neg T_{x3}\}$

$\neg T_{xi}$ is a tuple instance created such that T_{xi} and $\neg T_{xi}$ violate a functional dependency (FD). Pick 2 attributes A and B in the tuple. To inject an FD violation assign T_{xi} as $[i, \dots, a1, \dots, b1..]$ and $\neg T_{xi}$ as $[i, \dots, a1, \dots, b2..]$ where $a1$ is a value of the A attribute and $b1$ and $b2$ are values of the B attribute. The instances T_{xi} and $\neg T_{xi}$ thus violate the FD: $A \rightarrow B$.

(iii) Type is: $(\neg x_1 \vee \neg x_2 \vee x_3)$

We can also treat this as $(x_1 \wedge x_2) \rightarrow x_3$ Create the following tuple instances in U :

$\{\neg T_{x1}, \neg T_{x2}, T_{x3}\}$

(iv) Type is: $(\neg x_1 \vee \neg x_2 \vee \neg x_3)$

We can also treat this as $(x_1 \wedge x_2) \rightarrow \neg x_3$

Create the following tuple instances in U :

$\{\neg T_{x1}, \neg T_{x2}, T_y\}$

T_y is made such that T_y and T_{x3} violate the FD: $A \rightarrow B$.

At this point we have an instance of an FD consistency problem i.e., we have an uncertain relation U with uncertain

tuples and a single FD: $A \rightarrow B$ on this relation. This reduction, from the original 3-SAT problem has been done in time polynomial in the size of the original 3-SAT problem.

Our claim is that a solution to the 3-SAT problem exists iff there is a solution to the FD consistency problem we have derived. Say we have a solution to the FD consistency problem. Each tuple is some T_{xi} . For each i , we can have only one of T_{xi} or $\neg T_{xi}$ in the consistent relation obtained (so as to not violate the FD). For any T_{xi} in the solution we set x_i to 1 in the 3-SAT problem, for any $\neg T_{xi}$ in the solution we set x_i to 0. With this assignment we will necessarily have a truth assignment for the x_i s for which the 3-SAT formula is true. Also if there is a truth assignment that makes the 3-SAT formula true then there necessarily exists a solution to the FD consistency problem (for each x_i assigned to 1 we retain T_{xi} in the solution and for each x_i assigned to 0 we retain $\neg T_{xi}$). Conversely if there is no solution to the FD consistency problem then there is no solution to the 3-SAT problem.

Our claim above is thus valid that a solution to the 3-SAT problem exists iff there is a solution to the translated FD consistency problem. As 3-SAT is NP-complete it follows that the FD-consistency problem is NP-Hard.

Given the intractability of answering queries exactly in presence of ICs, we take a different approach that attempts to replace a given uncertain relation U by another *sub-relation* that is also (a special case of) an uncertain relation U' into which any constraints provided over U have been *factored in*. Ideally, U' represents all the possible worlds of U that are consistent w.r.t. C and eliminates possible worlds that are inconsistent. The uncertain relation shown in Fig 2 is such a U' for the relation U in Fig1. Answers to queries over U' , which can be efficiently evaluated using independence semantics, would thus be exactly the answers were we to execute the query over consistent possible worlds of U . In general, such a U' that *exactly* captures the set of consistent possible worlds of U might not exist. For instance if we modify the second tuple in the relation in Fig 1 to be

jim	manager	marketing (0.5)	MBA
		training(0.5)	

we can see that *no* sub-relation (in the or-set based model we use) can exactly represent the consistent possible worlds of U . Our goal, thus, is to identify a “good” sub-relation that mirrors/approximates the original uncertain relation (and constraints) closely. Such a “good” approximation would eliminate as many of the inconsistent worlds of the original relation as possible while at the same time minimizing the number of consistent worlds that would invariably be eliminated as a by product. The paper devises mechanisms to computing such a good approximation for the original uncertain relation given a set of integrity constraints (IC). We consider a large class of attribute, tuple, and relation level ICs - including FDs, aggregation constraints and other kinds of ICs that other approaches have not considered.

name	job-title	division	degree
jim	instructor (0.7)	training	MBA
jim	manager	marketing	MBA
jill (0.5)	consultant	innovation	PhD
jim (0.5)			

Fig. 2. Alternate Representation

Note that queries over a sub relation into which constraints have been factored can be answered efficiently. While simple selection queries over a single relation can be answered efficiently in a straightforward mechanism, techniques developed in [9] can be used to answer more complex single as well as multi relation queries. Our specific contributions can be summarized as: (i) We present a more general approach for factoring a large class of ICs into uncertain databases that other systems have not considered, (ii) By using approximations our approach can scalably handle uncertain databases with a high degree of data "dirtiness" (fraction of fields that are uncertain).

The rest of the paper is organized as follows. In Section 2, we formally define our notion of uncertain relations, state our problem of generating (tractable) sub-relations of uncertain relations as part of our approach to providing efficient retrieval over uncertain relations with constraints. Section 3 and 4 together develops our approach where we borrow from and build upon techniques from areas such as database repair [10] and work in compact representation of probabilistic distributions [1]. In Section 5, we demonstrate both the scalability and efficiency of our approach as well as impact of considering ICs on quality of the information extraction task. Section 6 gives an overview of related works and the last section concludes the paper.

II. FORMALIZATION

In this section we formally define uncertain relations with constraints and postulate the problem of generating approximations of such uncertain relations that facilitate efficient query answering.

Uncertain Relation An *uncertain relation*, U , is defined as:

- $U = \{t_1, t_2, \dots, t_n\}$; i.e., U is a relation which is a set of n tuples.
- $t_i = (a_{i1}, a_{i2}, \dots, a_{is})$; each tuple is a sequence of s attributes.
- $a_{ij} = \{(a_{ij}^1, c_{ij}^1), \dots, (a_{ij}^{k_{ij}}, c_{ij}^{k_{ij}})\}$; Each attribute is a set of possible attribute values with an associated probability distribution. The set is referred to as the attribute world. k_{ij} is the number of choices in the attribute world a_{ij} , and $\sum_{p=1}^{k_{ij}} c_{ij}^p = 1$.

Each uncertain relation U represents a set of *possible worlds*, PW_U . Each possible world corresponds to choosing a value for each attribute a_{ij} , a specific value from its attribute world. Let pw be a variable over the possible worlds. A possible world $pw = pw_q \in PW_U$ is defined by a function, $f_q : f_q(x, y) \rightarrow I$; where $x \in \{1, 2, \dots, n\}$, $y \in \{1, 2, \dots, s\}$ and $I \in \{1, 2, \dots, k_{xy}\}$. The number of such unique functions is $\prod_{i=1}^n \prod_{j=1}^s k_{ij}$ which is the number of possible worlds.

The probability distribution P_I defined over PW_U under the assumption of independence is :

$$\forall pw_q \in PW_U, P_I(pw = pw_q) = \prod_{i=1}^n \prod_{j=1}^s c_{ij}^{f_q(i,j)} \quad (1)$$

Note that $\sum_{all\ worlds\ q} P_I(pw = pw_q) = 1$.

The above model for representing database uncertainty is based on the *or-set* relations [11] where an attribute value is essentially a set of possible values with an associated probability distribution.

Uncertain Relation with Constraints We now associate constraints with uncertain relations, defining an uncertain relation *with constraints* denoted as $U + C$, where U is an uncertain relation as defined above and C is a set of integrity constraints over U . Let PW_U^C denote the subset of possible worlds in PW_U that are *consistent* w.r.t (*all*) the constraints, C . i.e., $PW_U^C = \{pw_q | pw_q \in PW_U \text{ and } pw_q \models C\}$. The set of possible worlds not consistent w.r.t. C is denoted as $PW_U^{-C} = \{pw_q | pw_q \in PW_U \text{ and } pw_q \not\models C\}$. The uncertain relation with constraints, $U + C$, is interpreted as a set of possible worlds of U with the probability distribution redefined as follows:

$$\begin{aligned} P(pw = pw_q) &= 0, \text{ if } pw_q \not\models C \\ P(pw = pw_q) &= P(pw = pw_q | pw \in PW_U^C), \text{ if } pw_q \models C \\ &= \frac{P(pw \in PW_U^C | pw_q) P_I(pw = pw_q)}{P(pw \in PW_U^C)} \\ &\quad (\text{Bayes' theorem}) \\ &= \gamma P_I(pw = pw_q) \end{aligned} \quad (2)$$

As $P(pw \in PW_U^C | pw_q) P(pw_q) = 1$ since $pw \models C$. Also $\gamma = 1/(1 - \lambda)$ where $\lambda = \sum_{pw_q \in PW_U^{-C}} P_I(pw_q)$.

Sub-relations Consider an uncertain relation U . If we replace the possible values of each attribute a_{ij} in each tuple t_i in U with a *subset* of the possible values for that attribute in U , we arrive at what we call a *sub-relation* of U . We denote the sub-relation of U by U' . Strictly speaking U' is not an uncertain relation as it does not necessarily provide a complete probabilistic distribution over possible relations. It is used however to represent a subset of the possible worlds for an uncertain relation. A sub-relation U' is additionally associated with a constant factor $\gamma_{U'}$ and the probability of any world $pw = pw_q \in PW_{U'}$ is given by $p(pw = pw_q) = \gamma_{U'} \prod_{i=1}^n \prod_{j=1}^s c_{ij}^{f_q(i,j)}$ i.e., the probability of any world is recalibrated with the $\gamma_{U'}$ factor. The factor $\gamma_{U'}$ is derived from Equation 2 which ensures that the probability of any *consistent* world in U' is exactly the same as in $U + C$. Note however that U' may represent some inconsistent worlds as well and assign a non-zero probability to such worlds.

As an example, Fig 3 represents a sub-relation of the uncertain relation in Fig 1 (and with the second tuple modified).

name	job-title	division	degree
jim	instructor(0.7)	training	BA
jim	manager	marketing (0.5) training(0.5)	MBA
jim (0.5) jill (0.5)	consultant	innovation	PhD

$\gamma = 1.17$

Fig. 3. Sub-relation

λ for the uncertain relation is 0.15. Thus $\gamma_{U'} = 1/0.85 = 1.17$ which is how the $\gamma_{U'}$ factor for the sub-relation in Fig 3 is derived. We define a sub-tuple of an uncertain tuple (any tuple in an uncertain relation is an uncertain tuple) analogously, where replacing the set of attribute values in each attribute in the tuple with one of its subsets provides us with a sub-tuple of that uncertain tuple.

We use sub-relations to approximate an uncertain relation U with constraints C . Ideally, we would like the sub-relation U' to represent the exact set of consistent possible worlds of U and to eliminate all of the inconsistent possible worlds. However, as discussed in the introduction, such a U' might not exist and, as a result, our goal will be to identify the "best" approximation of $U + C$. In order to define a concept of "best" we need to define a metric to evaluate how well does a specific sub-relation capture $U + C$.

Quality of Approximation: Let U be an uncertain relation with associated integrity constraints C and let U' be a sub-relation approximation of U . Let P_c be the (total) consistent mass in $U + C$ (i.e., the sum of the probabilities of the possible worlds of U that are consistent). Also, let C_r (I_r) be the consistent (inconsistent) mass of U retained in U' respectively. The quality of U' , denoted by $Q_{U'}$ is defined as:

$$Q_{U'} = \frac{C_r}{P_c} - I_r$$

Note that this metric considers the *absolute* inconsistent mass retained and the *relative* consistent mass retained because it is the fraction of consistent mass retained that we would like to be high (as opposed to its absolute value which may be low). A quality value of 1 is the best achievable. We also note that since the approximate representation U' might eliminate consistent possible worlds (in addition to eliminating inconsistent worlds), the results of a query Q over U' might include false negatives (i.e., tuples that should be part of the answer since they satisfy the query in some consistent world, but are not part of the result over U'). While introducing false negatives might be unacceptable for certain applications, for applications of probabilistic databases that motivate our work such as information extraction and query answering, we believe that a modest reduction in one of precision or recall in exchange for a significant increase in the other is a desirable tradeoff.

Problem Formalization Given the above definition of quality, we can now formally state our objective as that of generating a sub-relation U' of U that has the highest quality. That is, $\forall Y \in U'_M: Q_{U'} \geq Q_Y$, where U'_M is the

set of "all" sub-relations. Unfortunately, identifying such an "optimal" sub-relation is NP-hard even when we consider a single functional dependency or a tuple level constraint as we will see in the next section [12]. We will therefore restrict ourselves to heuristic techniques to finding U' that attempt to maximize $Q_{U'}$.

III. INCORPORATING ICS IN AN UNCERTAIN RELATION

In this section, we describe our approach to generating the approximate sub-relation U' given an uncertain relation U and a set of constraints C that hold over U . Our approach starts with the original relation U , selects a constraint $C_i \in C$, and attempts to resolve C_i by dropping (a minimal number) of attribute values from tuples in U such that the resulting sub-relation does not violate C_i . The process of resolving constraints (or "fixing" the relation U) is iteratively carried out until the algorithm deems that the benefit of further removing inconsistency no longer outweigh the loss of the consistent worlds that results as a by-product of "fixing" the uncertain relation. Before we discuss the details of the algorithm, we first need to specify the nature of integrity constraints (IC) that we consider in the paper. The approach we use to fix the uncertain relation depends upon the nature of the integrity constraint.

We classify ICs into the following three different categories:

(i) Attribute level ICs: Constraints that depend on the values of a specific attribute in a tuple, and not on other attributes in the same tuple or other tuples. An example can be `CHECK degreelevel(degree)` that states, through a user defined function (UDF), that the value for the degree must be at least a 4-year college degree. We will assume that attribute level ICs can be checked efficiently (in polynomial time).

(ii) Tuple level ICs: Constraints that are dependent on the values of two or more attributes within a specific tuple, and not on the values of attributes of different tuples. As an example: `CHECK compatible(job-title,degree)` may represent a tuple level IC that enforces, also through a UDF, some compatibility between a person's job title and his degree (e.g., that a "manager" must have at least an "MBA" degree, etc.). We will assume that each instance of a tuple-level IC can be checked for constraint violation efficiently (in polynomial time.) In addition, we will assume that the arity of the constraint, i.e. number of attributes associated with the constraint is small enough such that enumerating all tuple instances that could be potential constraint violations is tractable.

(iii) Relation level ICs: Constraints that exist across different attributes from different tuples. For instance a constraint: `CREATE ASSERTION no-multiple-divisions CHECK (SELECT COUNT division FROM employees GROUP BY (name, job-title) == 1)`

states that the same person cannot have the same job-title in two different divisions. This constraint is essentially the FD $(name, job-title) \rightarrow (division)$. Note that "Check" constraints at the attribute level (or at the tuple

Relation: U

name	job-title	division	deg
jim	instructor (0.7) manager (0.3)	training	BA (0.2) MBA (0.8)
jim	manager	marketing	MBA
jill (0.5) jim (0.5)	consultant	innovation	AAB (0.4) PhD (0.6)

Constraints: C

Attribute level ICs (C_a)

1. CHECK degreelevel(deg)

All employees have at least a 4 year college degree.

Tuple level ICs (C_t)

1. CHECK compatible(division,deg)

All "training" division employees have at least an "MBA" degree.

Relation level ICs (C_r)

1. CHECK (name, job-title) \rightarrow division

An employee does not hold the same title in 2 different divisions

jim	instructor (0.7) manager (0.3)	training	BA (0.2) MBA (0.8)
jim	manager	marketing	MBA
jill (0.5) jim (0.5)	consultant	innovation	PhD (0.6)

(a) U_1 : Factored attribute levels ICs

jim	instructor (0.7) manager (0.3)	training	MBA (0.8)
jim	manager	marketing	MBA
jill (0.5) jim (0.5)	consultant	innovation	PhD (0.6)

(b) U_2 : Factored tuple level ICs

jim	instructor (0.7)	training	MBA (0.8)
jim	manager	marketing	MBA
jill (0.5) jim (0.5)	consultant	innovation	PhD (0.6)

(c) U' : Factored relation level ICs, final approximation

TABLE I
GENERATING APPROXIMATIONS

Fig. 4. Uncertain Relation with Constraints

level) that depend upon other tuples will also be classified as relation level constraints.

The set of constraints, C , is a union of attribute level, tuple level, and relational level constraints, represented as C_a , C_t , and C_r respectively i.e., $C = C_a \cup C_t \cup C_r$.

We next discuss our strategies to resolve attribute, tuple, and relation level constraints independently. After describing our strategies to resolve single constraints, we will describe our algorithm to resolve the set of constraints C . In the remainder of the section, we will use the example uncertain relation with constraints in Figure 4 as an example for illustration.

Resolving attribute level ICs is actually trivial as in any attribute world we simply eliminate any attribute instance that is not consistent with an IC in C_a . This is illustrated in table I (a) where the **AAB** value in tuple 3 is dropped. We note that the sub-relation that results from resolving C_a removes only the inconsistent worlds but does not remove any consistent ones.

A. Resolving A Tuple Level IC

To resolve a tuple level constraint $C_{tup} \in C$, we can consider each tuple T of the uncertain relation independently. Given an uncertain tuple T and a specific tuple level constraint C_{tup} , we would ideally like to arrive at a sub-tuple T' (of T) that is equivalent to $T + C_{tup}$, i.e. it satisfies C_{tup} , while allowing the same set of possible consistent instances as T . Unlike the case of attribute level IC, dropping attribute values from tuples in U that violate C_{tup} might result in one or more consistent instances to be eliminated as well.

As a result, the resulting sub-relation U' might not exactly represent the set of consistent possible instances in $U + C_{tup}$. Fig 5 illustrates such an example with a constraint that all training division employees have at least an "MBA" level degree. Dropping any attribute value from the tuple results in a loss of a consistent instance. For instance, removing "BA" from the attribute world of degree attribute results in a sub-tuple that satisfies the considered tuple level IC, but it eliminates the consistent possible instance in which "jim" works in "marketing" division with a "BA" degree. Furthermore, the problem of identifying the sub-relation that optimally approximates (in terms of quality) the set of possible worlds of the uncertain relation U consistent w.r.t. a single tuple level constraint C_{tup} remains NP-hard. We state the following:

Statement 2: *Determining an optimal approximation T' of an uncertain tuple T is NP-Hard*

Proof: This follows by a reduction from the functional dependency (FD) consistency problem. The proof is as follows:

- 1) Consider any given instance of an FD consistency problem (U,F) where U is a U-relation and F is an FD over U.
- 2) Create a new tuple, T, as follows. For every tuple $t_i \in U$ create a new attribute A_{t_i} in T. For each tuple *instance* t_i^k in every tuple t_i in U, create a corresponding attribute value instance in A_{t_i} . Finally, provide a uniform probability distribution in all attribute worlds in T.
- 3) For every instance of a pair of tuple instances t_i^m and t_i^n ($i \neq j$) that violate F, create an instance of a constraint violation between the corresponding attribute value instances in T.
- 4) T is an uncertain tuple that possibly also has some constraint violations across attribute values. Note that the reduction from the FD consistency problem to this uncertain tuple T is done in time polynomial in the size of the original

name	job-title	division	degree
jim	instructor	training (0.6)	BA (0.7)
		marketing (0.4)	MBA (0.3)

Fig. 5. Sample U-tuple, for which no proper sub-tuple exists

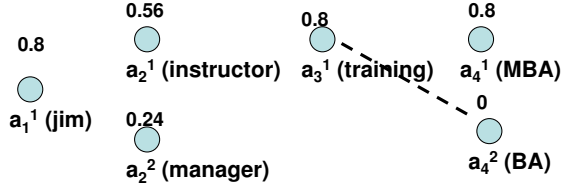


Fig. 6. Graph Representation of Uncertain Tuple

problem.

5) Generate an optimal approximation T' of T . If there is any tuple instance that is consistent in T then at least one such consistent instance *must* appear in T' . This is because all consistent tuple instances in T have the same probability and all inconsistent instances have a probability of 0. Also if T' is empty then this implies that there are no consistent tuple instances whatsoever in T .

6) The tuple instances in T directly correspond to relation instances in the original FD consistency problem as there is a 1-1 mapping from the attribute values instances in attributes in T to tuple instances in tuples in U . Any consistent tuple instance in T directly corresponds to a consistent *relation instance* in U .

7) The original problem of determining a consistent relation instance in U (or determining that none exists) is however NP-Hard. This implies that the problem of optimal tuple approximation, that this was reduced, to is also NP-Hard.

Given the intractability of identifying the optimal sub-relation, we focus on developing a heuristic approach to find a "good" approximation that preserves as much of consistent mass as possible (see Sec. 2) which we describe next.

Algorithm: APPLY_TUPLE_IC

Input: Uncertain relation U_0 , Tuple level IC C_{tup}

Output: Sub-relation U_1

```

1: APPLY_TUPLE_IC_SR ( $U_0, C_{tup}$ )
2:  $t_{new} \leftarrow \emptyset$ 
3: for (each tuple  $t$  in  $U_0$ )
4:    $t_{new} \leftarrow t_{new} \cup \text{APPLY\_TUPLE\_IC\_SR\_TUPLE}(t, C_{tup})$ 
5:  $U_1 \leftarrow \text{form\_relation}(t_{new})$ 
6: return  $U_1$ 

1: APPLY_TUPLE_IC_SR_TUPLE ( $T, C_{tup}$ )
2: ATTRIBUTE_MARGINALS( $T, S$ )
3:  $G \leftarrow \text{graph\_representation}(T, C_{tup})$ 
4:  $I \leftarrow \text{independent\_nodes}(G)$ 
5:  $N \leftarrow G - I$ 
6:  $Nb \leftarrow \text{best\_candidate\_to\_delete}(N)$ 

```

```

7:  $G \leftarrow \text{delete}(G, Nb)$ 
8:  $T \leftarrow \text{tuple\_representation}(G, \gamma)$ 
9: return( $T$ )

```

form_relation: Construct a new relation.

graph_representation: Convert uncertain tuple to graph.

independent_nodes: Find nodes without any edge.

best_candidate_to_delete: Find proper node to remove.

tuple_representation: Convert graph to tuple.

For a given tuple T of U and a tuple level constraint C_{tup} , we start with constructing a graph representation of T in which nodes correspond to attribute value instances in each attribute, and edges and hyper-edges represent sets of attribute value instances (across attributes) that violate the tuple level constraints. The graph representation of the first tuple in Fig 4 is shown in Fig 6. We now delete nodes in this graph till all the (hyper) edges disappear, the resulting graph represents the attribute value instances that are consistent w.r.t. C_{tup} and can hence be retained in the approximation. For choosing nodes to drop, recall that we are interested in approximations with high quality i.e., where any consistent mass dropped is minimal. The consistent mass associated with any individual node (attribute value) is given by its *marginal* probability in the tuple. The marginal probability of an attribute value instance a_{ij}^k , denoted as $p_{MARG}(a_{ij}^k)$ is defined as the sum of the probabilities of all the tuple instances implied by the uncertain tuple that include that attribute world instance.

$$p_{MARG}(a_{ij}^k) = \sum_{\text{all instances } t \in T \wedge a_{ij}^k \in t} p(t) \quad (3)$$

We adorn the graph nodes with their associated marginal probabilities. We then choose the nodes to drop in a greedy fashion biasing towards dropping nodes with low marginal probabilities, till all (hyper) edges have been eliminated. As an example, consider again the graph in figure 6, and its corresponding sub-tuple. Having just one tuple level IC, and a pair of violating possible attribute values, we can eliminate the only existing inconsistency, shown as the dashed edge in the graph, by dropping one of its corresponding nodes. In this case, we decide to drop a_4^2 , the "BA" value, according to its marginal probability, which is 0 comparing to the marginal probability of a_3^1 , "training" value, which is 0.8. The complete algorithm is described in Algorithm APPLY_TUPLE_IC.

Note that our approach requires that the marginal probability value for each attribute value instance in the tuple be known. Unfortunately, computing the marginal probabilities of attribute values instances in an uncertain tuple can be shown to be NP-Hard. Instead, we can *estimate* such marginals using statistical sampling. We employ naive-MC (Monte-Carlo) sampling. The procedure for estimating marginal probabilities of attribute value instances in a tuple, based on sampling randomly generated tuple instances, is described in algorithm ATTRIBUTE_MARGINALS.

Algorithm: ATTRIBUTE_MARGINALS

Input: Uncertain tuple T, Number of samples S

```
1: ATTRIBUTE_MARGINALS(T,S) {  
2:   for (all attribute value instances  $a_{ij}^k$  in all attributes in T)  
3:      $p_{MARG}(a_{ij}^k) \leftarrow 0$   
5:   for (i = 1 through S)  
6:      $t_{samp} \leftarrow \text{random\_sample}(T)$   
7:     for (all attribute value instances  $a_{ij}^k \in t_{samp}$ )  
8:        $p_{MARG}(a_{ij}^k) \leftarrow p_{MARG}(a_{ij}^k) + p(t_{samp})$   
10:  for (all attribute value instances  $a_{ij}^k \in T$ )  
11:     $p_{MARG}(a_{ij}^k) \leftarrow p_{MARG}(a_{ij}^k)/S$   
12:  return( $\{p_{MARG}(a_{ij}^k)\}$ )  
random_sample(T): random tuple instance.
```

Statement 3: *The derivation of marginal probabilities of attribute value instances in an uncertain tuple, or of tuple instances in an uncertain relation with constraints, is NP-Hard.*

Proof: Given an instance (U,F) of the FD consistency problem we determine the marginal probabilities of the tuple instances in each tuple in U. A consistent relation instance in U exists iff the marginal probability of at least one of the tuple instances (in any tuple in U) is >0 . The original FD consistency problem is however NP-Hard. This implies that determining the marginal probabilities of tuple instances in tuples in a U-relation is also NP-Hard

For determining the complexity of determining marginal probabilities of attribute values in an uncertain tuple we make a reduction from the FD consistency problem. Given an instance of the FD consistency problem (U,F) we create an uncertain tuple, T, exactly as in the proof for Statement 2 above. A consistent instance in the FD consistency problem is present iff the marginal probability of (at least) one of the attribute value instances in T is > 0 . The original FD consistency problem is however NP-Hard. This implies that determining the marginal probabilities of attribute value instance in attributes in an uncertain tuple is also NP-Hard.

B. Resolving A Relation Level IC

For a relation level IC the instances of violations of that IC could be exponential in the number of tuples. The approach we used for resolving tuple level ICs - which involves exhaustively enumerating and imprinting all instances of violations, is thus not practical for relation level ICs. Also in the context of a relation level IC, we will use the term "tuple instance" to refer to the projection of the tuple instances onto the attributes that are part of the IC. Resolving a specific relational level IC,

C_{rel} , in an uncertain relation U, comprises the following two steps:

a) Within U we identify *sets* of tuple instances where each set can potentially violate C_{rel} . For instance for a functional dependency IC, $A \rightarrow B$, where A and B are two sets of attributes according to the schema of U, any set of tuple instances which agree on the value of A, form a set of tuple instances that could potentially violate the FD. A possible relation of U, where tuple instances are drawn from such a set, could be inconsistent with C_{rel} . Given any C_{rel} , all such sets of tuple instances can be determined exhaustively (the number of such sets is proportional to the number of distinct attribute values of A). We refer to any such a set as a "NEED-FIX" class for that C_{rel} . As an example, a NEED-FIX class for the FD constraint over the uncertain relation in Fig 7 (a) is illustrated in Fig 7 (b). The tuple instances are denoted by first specifying the tuple number in the uncertain relation and then the tuple instance number within each tuple in ().

b) We eliminate the inconsistencies in any NEED-FIX class considering each class individually. This is achieved by *dropping* tuple instances in the class till consistency is achieved. We refer to this as "fixing" a class. For instance for a NEED-FIX class corresponding to a functional dependency $A \rightarrow B$, we would drop tuple instances until all the tuple instances in that class agree on the value of the attribute(s) in B. Note that in general there may be many different combinations of tuple instances that can be dropped that will achieve consistency. For instance, the NEED-FIX class in Fig 7 (b) can be fixed by dropping either the 1st tuple instance, or the 2nd and 3rd tuple instances in the class.

name	job-title	division
jim	instructor (0.5)	marketing
	consultant (0.5)	
jim	instructor (0.3)	training
	manager (0.7)	
jim	instructor	training

ICs:

1. CHECK (name, job-title) \rightarrow division
2. CHECK GROUP BY (name, job-title) COUNT * < 2

(a) Example uncertain relation with constraints

1(1) jim instructor marketing
2(1) jim instructor training
3(1) jim instructor training

(b) NEED-FIX class: FD IC

2(1) jim instructor training
3(1) jim instructor training

(c) NEED-FIX class: Aggregation IC

Fig. 7. An Example

The process of determining a NEED-FIX class, fixing it, and also the computational complexity of the fix operation are dependent on the *type* of the relational constraint that is being addressed. We described the generation and fixing of NEED-FIX classes for FD type ICs above. For aggregation constraints, such as the 2nd IC in the example in Fig 7 (a), NEED-FIX classes are determined by grouping together tuple instances that agree on the attributes that we must *group* by

CONSTRAINT TYPE	Generating NEED-FIX class(NF)	Fixing NF	Complexity*
<i>Type:</i> Functional Dependency (FD) <i>Format:</i> $A \rightarrow B$ where A,B are subsets of columns in U	1) For each tuple instance t in each tuple T in U. 2) Initialize a new NEED-FIX class, NF, with the single member t. 3) For any tuple T' in U that contains a tuple instance t' such that $t'.A=t.A$, add t' to NF. 4) Add NF to the pool of NEED-FIX classes.	1) Group the tuple instances by the value of B 2) Select the value for B for which the sum of the marginals (of the tuple instances) in that group is the highest. 3) Drop all tuple instances with values for B other than the above selected value.	$O(N_t)$
<i>Type:</i> Inclusion Dependency(IND) <i>Format:</i> $U.A \in E.B$ where E is a fixed relation and A, B are subsets of columns in U and E respectively	1) Initialize a new NEED-FIX class,NF, to NULL. 2) For any tuple instance t in tuple T, if $t.A \in E.B$ then add T to NF. 3) Add NF to the pool of NEED-FIX classes.	1) Drop all tuple instances in NF.	$O(N_t)$
<i>Type:</i> Aggregation <i>Format:</i> GROUP BY A COUNT < G ;where A is a subset of columns in U and G is an integer.	1) For each tuple instance t in each tuple T in U. 2) Initialize a new NEED-FIX class,NF, with the single member t. 3) For any tuple T' in U that contains a tuple instance t' such that $t'.A=t.A$, add t' to NF. 4) Add NF to the pool of NEED-FIX classes.	1) Let Nnf be the number of tuple instances in NF. 2) If $Nnf < G$ then we are done. 3) Else $Nnf - G$ tuple instances have to be dropped. Drop those $Nnf - G$ tuple instances from NF for which the sum of the marginal values is minimum.	$O(N^T C_{N_T G})$
<i>Type:</i> Aggregation <i>Format:</i> GROUP BY A EXP(B) θ val; where EXP is one of {AVERAGE, SUM, COUNT}, A is a subset of columns in U, and B is a (numeric) column in U, and θ is one of {=, <, >}	Same as above.	1) Exhaustively search all combinations of tuple instances that can be dropped to make NF consistent wrt this constraint. 2) Determine the combination with the minimum total marginal value and drop the tuple instances in that combination.	$O((2^P)^{N_T})$
<i>Type:</i> SET Constraint <i>Format:</i> $Q \theta E.B$; where $Q = (SELECT A FROM U WHERE CND)$, CND is a query condition, and θ is one of {=, <, >}	1) Initialize a new NEED-FIX class,NF, to NULL. 2) For any tuple instance t in tuple T in the result of Q, if $t.A \in E.B$ then add T to NF. 3) Add NF to the pool of NEED-FIX classes.	1) Drop all tuple instances in NF.	$O(N_t)$

TABLE II

ADDRESSING RELATION LEVEL CONSTRAINTS (N_t : TOTAL NUMBER OF TUPLE INSTANCES IN NF; N_T : TOTAL NUMBER OF TUPLES REPRESENTED IN NF; P : MAXIMUM NUMBER OF TUPLE INSTANCES IN ANY TUPLE IN NF.)

according to the aggregation constraint. One such NEED-FIX class is shown in Fig 7 (c) where we have grouped together tuple instances by (name, job-title). The fix is a process of eliminating tuple instances such that the aggregation constraint *condition* is satisfied, in this example dropping either of the tuple instances in the NEED-FIX class will ensure this. Table II presents the specific kinds of relation ICs addressed and the associated complexity. The procedures for generating and fixing NEED-FIX classes for IND and SET constraints are straightforward and we do not present them here for lack of space. As we have seen there can be multiple sets of tuple instances that can be dropped to fix a NEED-FIX class. The choice of an optimal set of tuple instances to

drop is made based on the marginal probabilities of each tuple instance. Formally, the marginal probability, $p_{MARG}(t)$, of a tuple instance, t in an uncertain relation U is defined as:

$p_{MARG}(t) = \sum_{all\ instances\ u \in U} p(u)$; $t \in u$. Like attribute marginals, the derivation of tuple instance marginals in a relation is also NP- Hard [12]. We employ naive-MC sampling for estimating these marginals in a manner analogous to the attribute marginals estimation, and here we sample randomly generated relation instances.

For any NEED-FIX class we can determine the combination of tuple instances with lowest (total) marginal probability, that if dropped will eliminate the inconsistencies in that class. The complexity of resolving a NEED-FIX class is

polynomial in the size of the NEED-FIX class for (the permitted) FDs, INDs and SET constraints and is exponential (in the size of the NEED-FIX class) for the aggregation constraints. For aggregation constraints we use a simple hill-climbing procedure to find a set of tuple instances to drop that will remove the inconsistency in the NEED-FIX class and also have a low (total) marginal probability.

Algorithm: APPLY_RELATION_IC

Input: Uncertain relation U_0 , Relation level IC C_{rel}

Output: Sub-relation U_1

```

1: APPLY_RELATION_IC( $U_0, C_{rel}$ )
2:  $NF \leftarrow \emptyset$ 
3:  $\gamma \leftarrow \text{estimate\_gamma}(U_0, C_{rel})$ 
4: for (each tuple instance  $t$  in each tuple  $T$  in  $U_0$ ) {
5:    $NF_t \leftarrow \text{generate\_need\_fix\_class}(t, U_0, C_{rel})$ 
6:    $NF \leftarrow NF \cup NF_t$ 
7: }
8:  $NF \leftarrow \text{FIX}(NF)$ 
9:  $U_1 \leftarrow \text{form\_relation}(NF, \gamma)$ 
10: return  $U_1$ 

```

generate_need_fix_class: generates a new NEED-FIX class given a tuple instance and a relation level IC.

fix: fix a particular NEED-FIX class

form_relation: form a new relation.

IV. USING A MULTI-ROW REPRESENTATION

Revisiting the example in Fig 5 we realized that in order to achieve consistency (by dropping some instances) some loss of consistent instances was invariable. This is because of the model simplicity and we have been using what is called a single-row model [1]. A representation model that permits multiple rows for each tuple, known as a multi-row model, can overcome this limitation as illustrated in Fig 8 where the approximation now exactly captures the uncertain tuple in Fig 5.

jim	instructor (0.7) manager (0.3)	training (0.6) marketing (0.4)	MBA (0.8) 1
jim	instructor (0.7) manager (0.3)	marketing (0.4)	MBA (0.8) 1 BA (0.2)

Fig. 8. Uncertain Tuple Approximation

While in the above example the multi-row representation exactly captured the uncertain tuple with a small number of rows (2), this is not the case in general. We present the following:

Statement: *The number of rows in a multi-row representation required to exactly capture an uncertain tuple with constraints can be exponential (in the size of the largest attribute world*

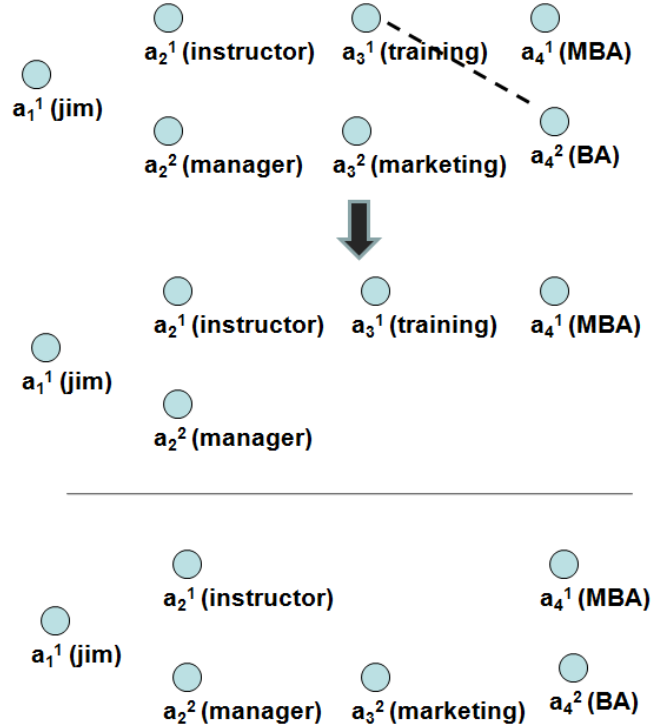


Fig. 9. Multi-Row Example

in the tuple) in the worst case.

Proof: Given a tuple and a set of constraint violations (let us consider only binary constraints violations across pairs of attribute values wlog) assume that there is a multi-row representation with M rows. Consider any row, r , where we have at least one attribute that has at least 2 attribute values. We *insert* a new violation between any of these (multiple) attribute values and any attribute value in any other attribute in the tuple. Now r must necessarily be split into at least 2 rows to exactly capture the consistent tuple instances. We can continue inserting violations in rows in this manner with an upper bound of KC_2A^2 violations that we will insert where K is the number of attributes. The number of rows that we will form in the multi-row model can however be as much as $O(A_K)$ i.e., exponential in the (maximum) size of the attribute worlds in the tuple.

An approximation that takes exponential space is not tractable to reason with and we are interested in multi-row approximations where the number of rows is bounded by a constant or at least a factor that is polynomial in the size of the original uncertain tuple. With such a restriction we can at best achieve an optimal approximation as opposed to an exact one in the general case. Any multi-row approximation is defined by 2 kinds of *parameters*, one is the number of rows in the representation and the other is the assignment of probabilistic values to attribute value instances within each attribute within each row. The complexity of deriving an optimal approximation is an issue however, we present the following:

Statement: For a multi-row representation with a bounded number of rows, determining multi-row model parameters that result in an optimal approximation of a tuple is NP-Hard.

Proof: This too follows from a reduction from the FD consistency problem, and the proof is analogous to as for the single row model.

Algorithm: APPLY_TUPLE_IC_MR

Input: Uncertain relation U_0 , Tuple level IC C_{tup}

Output: Sub-relation U_1

```

1: APPLY_TUPLE_IC_MR ( $U_0, C_{tup}$ )
2:  $t_{new} \leftarrow \emptyset$ 
3: for (each tuple  $t$  in  $U_0$ )
4:    $t_{new} \leftarrow t_{new} \cup \text{APPLY\_TUPLE\_IC\_MR\_TUPLE}(t, C_{tup})$ 
5:  $U_1 \leftarrow \text{form\_relation}(t_{new})$ 
6: return  $U_1$ 

```

```

1: APPLY_TUPLE_IC_MR_TUPLE( $t, C_{tup}, M$ )
2:  $V \leftarrow \text{determine\_violation\_sets}(T)$ 
3:  $m=0, F=0$  4: while ( $m < M$  and inconsistent( $T$ ))
6:    $TopV \leftarrow \text{top\_violation\_set}(V)$ 
7:    $T \leftarrow \text{split}(T, TopV)$ 
8: end while
9: for (each row  $R \in T$ )
10:   $R \leftarrow \text{APPLY\_TUPLE\_IC\_SR}(R)$ 
11: end for
12: return  $T$ 

```

What we employ is a heuristic approach to generating a multi-row approximation for a given uncertain tuple. We describe our approach using the example of binary tuple level constraints although the basic approach is valid for general (k-ary) tuple level constraints. Continuing with the graph representation of a tuple as described earlier, we recall that our aim was to eliminate (hyper) edges in the graph by dropping nodes. In the multi-row model our aim is to instead *split* the graph into multiple sub-graphs such that the (hyper) edges are eliminated - this is illustrated in Fig 9 where the original tuple graph is split into two sub-graphs neither of which contains the edge. The idea is to split a tuple graph recursively in this manner till (i) No sub-graph contains any edges, or (ii) The number of sub-graphs exceeds the number of available rows per tuple - whichever is earlier. Each sub-graph then corresponds to a row in the multi-row representation of the tuple. Consider an uncertain tuple T and three of its attributes A_i, A_j , and A_m with attribute value instances as shown in fig 10. Focusing on attributes A_i and A_j , certain attribute value instances in A_i may be inconsistent with certain instances in A_j , based on 1 or more (binary) tuple level constraints. For an attribute value instance $a_{ik} \in A_i$, define $\text{cons}(a_{ik}, A_j)$ as the set of those attribute value instances in A_j that are consistent with a_{ik} i.e., wrt the

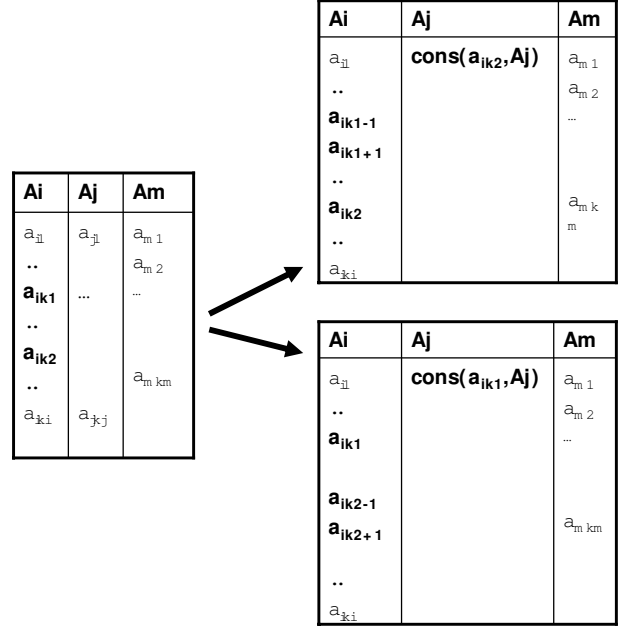


Fig. 10. Split to Multi-Row

tuple level constraints. Now consider a particular row in a multi-row representation for T . A row is said to be consistent wrt attributes A_i and A_j iff all attribute value instances for attribute A_i in that row are consistent with all attribute value instances for A_j in that row. A row is said to be completely consistent if it is consistent wrt all pairs of attributes in the uncertain tuple. A row is inconsistent if it is not consistent wrt at least one pair of attributes A_i and A_j . It follows that any row will be inconsistent iff there are 2 attributes A_i and A_j such that there are two instances in A_i , a_{ik1} and a_{ik2} and $\text{cons}(a_{ik1}, A_j) \neq \text{cons}(a_{ik2}, A_j)$. We denote any such pair of attribute value instances and pair of attributes where this an inconsistency as a *violation set* $v = \langle a_{ik1}, a_{ik2}, A_j \rangle$.

To eliminate the inconsistencies across A_i and A_j , the strategy we follow, in the single-row model, is to eliminate certain attribute value instances from attributes A_i and/or A_j till consistency is achieved. This comes at a cost of possibly eliminating certain consistent instances as well and we provided a mechanism to estimate this loss using marginal values in the previous section. We denote as $\text{loss}(v)$ the estimate of the consistent mass loss associated with making violation set v consistent by eliminating attribute value instances. With the luxury of multiple row, we can instead split a row with a violation set into 2 rows as shown in fig 10. The resulting 2 rows are necessarily consistent wrt A_i and A_j . Denote this operation as that of splitting on a violation set. Note that the inconsistencies are eliminated but no consistent mass is lost in the process. We can perform such splitting on all violation sets for the uncertain tuple, the number of violation sets is polynomial in the (maximum) number of attribute value instances in each attribute and the number of attributes. While this will ensure that we end up in a multi-

row representation that exactly captures all the consistent tuple instances in the original uncertain tuple, the number of rows created can be exponential. The number of rows however is bounded. Prioritizing and considering violation sets, based on decreasing order of $loss(v)$, we split them till either all inconsistencies are eliminated or we reach the limit of the number of rows, whichever is earlier. Should the limit on number of rows be reached first there will be rows that do have inconsistencies (still) present. We employ the single-row approximation on each of these rows. The heuristic rationale is that the additional row created due to splitting is in a sense saving us the associated loss value of consistent mass.

V. INTEGRITY CONSTRAINT SELECTION

In the previous section we studied how individual ICs of different kinds can be applied to remove inconsistency in an uncertain relation with constraints. Strictly speaking, when we state we are resolving a tuple (relation) IC we mean we are resolving that tuple (relation) IC in a particular tuple (NEED-FIX) class that is inconsistent with that IC. This is what the term "resolving an IC" will imply now on. In this section we describe how a set of ICs can be applied so as to achieve an approximation U' of good quality. Note that if our goal was to simply eliminate all the inconsistency we could apply all the ICs and achieve this, however we realize that a significant amount of consistent instances can be lost this way. The challenge is to find an optimal *subset* of ICs to apply such that the quality of the approximation achieved is maximized.

A. Utility of Each IC

For each *individual* IC we can determine whether resolving it will cause the overall quality to increase or decrease. Assume that for any IC we have an estimate of the inconsistent mass lost, IC_L , and the consistent mass lost, CM_L , as a result of applying that IC. We define the *utility* of an IC, UT , as $UT = IC_L - CM_L/P_c$, where P_c is the total consistent mass in the uncertain relation. The reader can verify, given the quality measure definition in Section 2, that the overall quality will necessarily increase after resolving that IC if its utility UT is > 0 .

We need to be able to determine the utility for any IC. Recall that a tuple inconsistent with a tuple level IC can be resolved by dropping a single attribute value in some attribute (involved in the IC violation). IC_L in this case can be determined by computing the probability of the tuple instances in the tuple that are indeed inconsistent w.r.t. that IC. CM_L on the other hand is nothing but the marginal probability of the attribute value instance that we will drop. Determining UT for a relation level IC is relatively more complicated. Recall that resolving a tuple IC in each NEED-FIX class is a process of eliminating attribute values in possibly multiple tuples. IC_L is determined by statistical sampling within a NEED-FIX class i.e., by randomly generating relation possibilities from the NEED-FIX class and estimating what is inconsistent. Now to fix the class if the attribute values to be dropped (across different tuples) are av_1, \dots, av_n then $P(av_1 \cup \dots \cup av_n)$

is a measure of the consistent mass lost by dropping these attributes. This is essentially the estimation of a DNF formula which can be also be done using Monte-Carlo sampling and applying the Luby-Karp-Madras estimation algorithm [13].

Algorithm: Greedy_IC_Resolution

Input: Uncertain relation U , Set of ICs C , Threshold B

Output: Sub-relation U'

```

1:  $C_t \leftarrow C$ 
2:  $U_t \leftarrow U$ 
3:  $c_m \leftarrow null$ 
4: initialize_utilities( $C_t$ )
5: while ( $C_r(U_t) > B$  and  $C_t \neq \emptyset$ )
6: do
7:   UPDATE_UTILITIES( $U_t, C_t$ )
8:    $c_m \leftarrow \text{select\_best\_IC}(C_t)$ 
9:    $U_t \leftarrow \text{resolve}(U_t, c_m)$ 
10:   $C_t \leftarrow C_t - c_m$ 
11: end
12: return  $U_t$ 
```

```

1: UPDATE_UTILITIES( $U_t, C_t$ )
2: for (each IC  $c_i$  in  $C_t$ )
3:    $\text{benefit}(c_i) \leftarrow \text{calculate\_benefit}(c_i, U_t)$ 
4:    $\text{cost}(c_i) \leftarrow \text{calculate\_cost}(c_i, U_t)$ 
5:    $\text{utility}(c_i) \leftarrow \text{benefit}(c_i) - \text{cost}(c_i)$ 
```

initialize_utilities: Define a utility value for each IC initialized with unknown

select_best_IC: Select the IC with the maximum utility
resolve: Resolve given IC in the sub-relation according to its type

calculate_benefit: Calculate benefit of given IC, if resolved in the sub-relation

calculate_cost: Calculate cost of given IC, if resolved in the sub-relation

B. IC Selection

Based on the utility, we need to determine an optimal set of ICs to choose to arrive at a maximum quality approximation. The complexity in this problem is caused by the fact that there can be *shared dependencies* amongst the resolution for certain ICs, specifically this happens if some of the attribute values to be dropped are common across multiple ICs. The utility of applying a set of multiple ICs thus cannot be determined from the utilities of the individual ICs alone. The problem of determining a subset of ICs that maximizes the resulting approximation quality, where costs and benefits may be shared across the ICs can in fact be restated as the Budgeted Maximum Coverage (BMC) problem [14], which unfortunately is NP-Hard. We thus provide a heuristic algorithm that attempts to find a subset of ICs to apply such that we achieve an approximation of high (although not necessarily the highest) quality.

Our approach is to first consider all tuple level ICs and associated tuples and resolve them (or not). We then move on to considering relation ICs and associated NEED-FIX classes. Within each of the two categories of ICs we consider and resolve an IC and an associated tuple or NEED-FIX class in a *greedy* fashion. The algorithm selects ICs (and tuples or NEED-FIX classes) in descending order of the associated utility. After each iteration, the utilities of each of the ICs (and tuples or NEED-FIX classes) are recalibrated. This is to factor in the attribute value instances that have already been dropped as a result of the ICs that have so far been applied. The algorithm applies ICs sequentially in this manner, recomputes utilities at each iteration, and terminates when we have no more ICs with an associated utility that is > 0 . Greedy_IC_Resolution describes this algorithm.

Estimation of Key Quantities In the above approximation and IC selection algorithms we require the value Pc - total consistent mass in an uncertain relation, Cr - consistent mass retained and Ir - inconsistent mass retained for any approximation U' . Determining any of these values is also NP-Hard. We state:

Statement 4: *The derivation of the total consistent mass δ (or $\gamma = 1/\delta$) factor for a U-relation with constraints is NP-Hard*

Proof: Given an instance of the FD consistency problem, consider the δ factor for the uncertain relation U in that problem. A consistent relation instance in the original FD consistency problem is present iff δ , the total consistent mass is > 0 . This implies that if δ (or γ) can be determined in polynomial time, then the FD consistency problem can be addressed in polynomial time as well. As the original FD consistency problem is NP-Hard, it follows that determining the δ (or γ) factor for a U-relation is also NP-Hard.

We thus resort to statistical sampling to estimate these values instead. A naive approach however is not applicable in this case. Consider estimating Pc given an uncertain relation U . We can estimate the *average* consistent mass per world instance, Pc_{AVG} , and then multiply this by the number of world instances (which we can compute directly). We recall Hoeffding's inequality [15] from basic probability theory which states:

Hoeffding's Inequality: Let X_1, X_2, \dots, X_n be iid random variables, while for all i we have $a_i \leq X_i \leq b_i$, and also let $S = \sum_{i=1}^n X_i$. Then we have:

$$Pr(S - E[S] \geq nt) \leq e^{(-2nt^2) / \sum_{i=1}^n (b_i - a_i)^2} \quad (4)$$

Or: $Pr(SAvgX - EAvgX \geq t) \leq e^{(-2nt^2) / \sum_{i=1}^n (b_i - a_i)^2}$ where $SAvgX = (\sum_{i=1}^n X_i) / n$ is the sample average and $EAvgX$ is the expected average of the X_i s.

Treating the mass of a single world instance as a random variable X_i above the sample average $SAvgX$ is an estimate of Pc_{AVG} . The value t is a measure of the error. To estimate a small quantity such as Pc_{AVG} which for an uncertain relation with 3 attributes, 2 attribute values per attribute, and 100 tuples is itself of the order of 2^{-300} , to within say a 10% error requires t to be accordingly small as well. Plugging such a

jim	instructor	training (0.6)	MBA (0.3) 1
jim	instructor (0.3)	marketing (0.4)	BA (0.7) 1
			MBA (0.3)

Fig. 11. Multi-Row Representation

small value of t and using 0 and 1 as lower and upper bounds for X_i we see that we require an extremely large number of samples (order of 10^{30}) n to achieve a probability of 0.9 that the estimation error is within 10%. Instead of the average consistent mass we estimate the *ratio*, R , of the consistent mass to the total mass. We define a *block* in U (or U') to be any subset of relation instances from the possible world of U (or U'). For any such block B_i define the quantity:

$$R_{B_i} = \text{Total consistent mass in } B_i / \text{Total mass in } B_i$$

We choose block size for a block B_i such that R_{B_i} can be computed by exhaustively enumerating through all instances in that block. The average value of R_{B_i} , referred to as AvgR, is simply $\sum_{i=1}^N R_{B_i} / N$. Unlike Pc_{AVG} or Cr_{AVG} , AvgR is in general not such an infinitesimally small quantity (for instance 0.3 could be a value of AvgR). Thus the number of samples required to estimate AvgR to within a reasonable accuracy is significantly smaller, for instance a confidence of 0.9 of estimating this to within 10% error would require sampling just a few hundred such blocks (Equation (4)). Now for both U or an approximation U' we can determine the total mass. For U it is simply 1, and for U' we can just compute it given U' . Having the total mass, and a reasonable estimate of AvgR we can derive reasonably accurate estimates of Pc , or Cr and Ir .

VI. OTHER ISSUES

While we have described the basic approach to resolving various kinds of ICs we would like to discuss some additional issues related to the representation model and the IC resolution algorithms.

Model Expressivity The or-set model we have used is simple and efficient but also limited in expressivity. With more expressive models we will achieve better quality approximations as this will mean having to drop less consistent mass. We have begun exploring more expressive models with using a *multi-row* representation for tuples where a tuple can be represented as multiple rows of or-sets of attributes. This is illustrated in Fig 11 where we note that we can now exactly represent the consistent instances of tuple of Fig 5. Our experimental results also show that we achieve better quality approximations using multiple rows. Developing an approach for approximating an uncertain relation with constraints to a more complex model such as that based on world set decompositions and "ws-sets" [5] is indeed an interesting direction for future work.

Incrementality While in most applications we expect the complete uncertain relation and set of ICs to be provided upfront, we can also envision scenarios where the additions to the ICs, to the uncertain relation itself (i.e., new tuples), or both, are provided incrementally. Rather than recompute U' from scratch in such cases, we present an incremental

approach. Consider first the case where we have approximated an uncertain relation U to U' given a set of ICs, and are now given a new set of additional ICs $C_{Na} \cup C_{Nt} \cup C_{Nr}$, where C_{Na} , C_{Nt} , C_{Nr} are the additional attribute, tuple, and relation level ICs respectively. Our approach is to start with U' , apply the additional attribute ICs, and then apply the additional tuple and relation ICs in a greedy fashion using the algorithm GREEDY_IC_Resolution. The steps are as follows:

1. Resolve C_{Na} in U' resulting in U_1'
2. Resolve $C_{Nt} \cup C_{Nr}$ in a greedy fashion on U_1' , resulting in U_2' which is now the new approximation of U .

The other case is when new tuples are provided for U . Let the set of new tuples be U_N . As attribute and tuple level ICs are local to individual tuples we need resolve the (existing) attribute and tuple level ICs *only* in U_N . New violations of relation level ICs however can occur within the tuples in U_N or across the tuple in U_N and U' . We thus proceed as follows.

1. Resolve C_{Na} in U_N resulting in U_{N1} .
2. Resolve C_{Nt} in a greedy fashion on U_{N1} resulting in U_{N2} .
3. Resolve C_{Nr} in a greedy fashion on $U' \cup U_{N2}$, resulting in U_1' which is now the new approximation of U .

Operations and ICs We achieve consistency with the ICs by essentially deleting tuples (the deletion of an attribute value instance can be viewed as deleting the tuple instances that get dropped as a consequence). In database *repair* one can in general consider any of tuple deletion, addition, or modification to repair a database to make it consistent with a set of given ICs. Our model is to start with a *complete* uncertain relation i.e., one where we know of *all* the possible relations that that uncertain relation implies. Starting with the complete space of possibilities, the only meaningful operation to ensure consistency given ICs, is to eliminate possible relations that are inconsistent with any of the ICs. Coming back to a repair perspective, the deletion of tuples is the only viable option in this framework. Another related aspect is that we permit only particular subtypes of ICs within the classes of relation ICs addressed as shown in Table II. This is to ensure that a NEED-FIX class wrt these kinds of constraints can *always* be fixed using tuple deletion.

VII. EXPERIMENTAL EVALUATION

We present experimental evaluation results in two different experimental set-ups. The first set-up is to assess the impact of incorporating constraints on applications that use uncertain relations - specifically we choose the application of information extraction, and assess an eventual improvement in extraction accuracy with the use of constraints. This experiment is over a real dataset of free text bios of researchers collected from their homepages on the open Web. The second set-up is to evaluate the effectiveness of our approach for approximating an uncertain relation with constraints and our primary goal is to assess the quality of the approximations achieved. We

employ a synthetic dataset in this case. We describe below the two sets of experiments and results.

A. Application Impact

We consider the application of information extraction (IE), in particular the task of "slot-filling" or extracting relations from text. Our goal is to assess any improvement in extraction accuracy that can be achieved with the use of ICs. We store the extracted data provided by a given extractor in an uncertain relation. We further define a set of ICs that are meaningful for the particular relation that is being extracted. We then compare the accuracy of retrieval done over the original uncertain relation, with the uncertain relation refined incorporating the ICs.

1) *Dataset*: We have chosen the extraction task of extracting details of a researcher, such as her job-title, employer, academic degrees and their associated dates and alma-maters from *free text* bios on their Web pages. We have collected around 500 such Web pages of bios from the homepages of researchers in the field of computer science. We identified 48 different items or slots to be extracted from each Web page which correspond to the above mentioned data items such as degrees, dates, employers etc.

2) *Uncertain Relation Representation*: We then trained and employed the TIES [16] information extraction system to extract these slots from the collection of Web pages. The extracted data is first represented in an uncertain relation. We consider each Web page as providing the data for a single tuple in this relation. State-of-the-art extraction systems such as TIES now provide a *space* of multiple possibilities for an extracted value for a slot, typically having each possible value associated with a confidence score. The extracted values provided by the extractor for a particular slot are part of the space of attribute values for the corresponding attribute and tuple in the uncertain relation. Also other possible values for that slot, identified through a *tokenization* process are included in the space of possible attribute values, realizing a complete space of attribute value possibilities. As an example, for a particular page (tuple) say the extractor returns the set of values [(2005 9.2) (2001 1.3)] for the PhD Date attribute i.e., two possible values and associated confidence scores. Also suppose that through tokenization we know that one other token, (2003), which is also of the type date (which is the domain for the PhD Date attribute), *could also* be a value for that slot. The attribute world formed based on this information is {(2005 0.6), (2001, 0.1), (2003, 0.3)}. (The details such as the determination of the probabilistic distribution in each attribute world are important in general, but not to this discussion). The set of attribute worlds corresponding to all slots for a page forms an uncertain tuple and the set of all such tuples (corresponding to all pages) corresponds to the extracted uncertain relation that we will call U_{bios} .

3) *Integrity Constraints*: Next, we author a set of integrity constraints that capture the semantics of the bios relation. For instance we know that people receive their PhD degrees only *after* their bachelors degrees (in the same major at least), or

Slot	p_i	p_c	r_i	r_c	f_i	f_c
Title	0.95	0.8	0.78	0.94	0.85	0.82
Employer	0.79	0.82	0.65	0.69	0.71	0.75
PhD Degree	0.98	0.98	1	1	0.98	0.98
PhD School	0.69	0.76	0.36	0.58	0.47	0.66
PhD Date	0.69	0.86	0.46	0.83	0.55	0.84
Bach School	0.93	0.9	0.3	0.49	0.45	0.63
Bach Date	0.88	1	0.62	0.96	0.73	0.98

TABLE III
EXTRACTION ACCURACY WITH CONSTRAINTS

we know that a person who received his PhD in 1978 is not likely to have a current job title of an Assistant Professor. For this domain we were able to specify a total of over 40 ICs spanning the attribute, tuple, and relation levels. A subset of such constraints are: 1) *All computer science degrees were awarded after 1959.* 2) *A person receives his doctoral degree only after his bachelors degree (same major).* 3) *A NULL value for a degree implies NULL values for the associated alma-mater and degree date.* 4) *The PhD degree alma mater and employer of a person are different.* The first constraint above can be expressed as an attribute level IC while the other 3 can be expressed as tuple ICs over U_{bios} . Strictly speaking, some of the above constraints (such as 4) are "soft" constraints in that they hold mostly but not necessarily always. For our purpose we treat them as hard constraints.

4) *Results:* We evaluated the precision and recall of retrieval over several different slots in U_{bios} . We compare the accuracy of retrieval over the original extracted uncertain relation U_{bios} , with that over U_{bios} augmented with the domain ICs. We consider precision and recall on a per-slot basis, where:

Precision for a slot s , $PR(s)$, is defined as:

$$PR(S) = \sum_{\text{all tuples } t} p(v)_{s,t}/N \quad (5)$$

where v is the correct value for the slot s in tuple t , $p(v)_{s,t}$ is the probability associated with value v for slot s in tuple t , and N is the number of tuples returned.

Recall for a slot s , $RE(s)$, is defined as:

$$PR(S) = \sum_{\text{all tuples retrieved } r} p(v)_{s,r} / \sum_{\text{all tuples } t} p(v)_{s,t} \quad (6)$$

where v is the value for slot s in tuple t .

Given U_{bios} and the set of ICs specified over this relation we generate an approximation of U_{bios} plus the ICs, U_{bios}' using our approach. Table 3 provides the retrieval accuracy, in terms of precision, recall, and f-measure, for a subset (for brevity) of the slots over both U_{bios} and U_{bios}' . Here p_i , r_i , and f_i are precision, recall and f-measure respectively over U_{bios} , and p_c , r_c , and f_c are the corresponding values over U_{bios}' . We observe that both precision and recall for many slots are significantly improved in U_{bios}' compared to U_{bios} , thus demonstrating the effectiveness of employing ICs. Albeit in some cases we see a (minor) drop in precision which

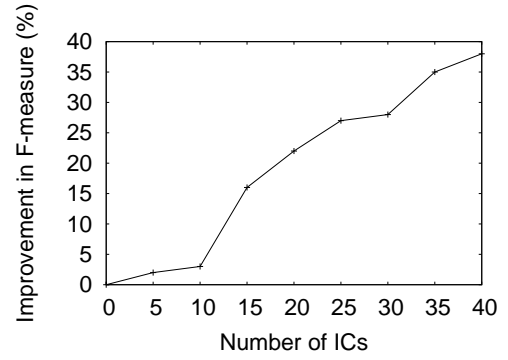


Fig. 12. Extraction Accuracy

Param	Description
A	Number of attributes in relation.
MAX	Maximum number of choices in one attribute
C	Total Number of ICs
D	Maximum arity of a (tuple) IC
R	Number of tuples
α	Degree of data dirtiness (% fields uncertain)

Fig. 13. Synthetic Data Generator Parameters

is due to treating what should be soft constraints as hard. Note that these are extraction accuracy improvements over the output of extraction systems that are representative of the state of the art and also have been provided extensive training data in the application domain. These results demonstrate the utility of employing constraints in the context of an actual application of information extraction where the use of constraints significantly improves the retrieval quality. We also demonstrate (in Fig 12) the increase in overall extraction accuracy (aggregated over all the slots in the relation) as a function of the number of ICs incorporated.

B. Assessing Effectiveness of Approximation Approach

Our aim is to arrive at a good quality approximation of an uncertain relation with constraints. For more detailed analysis of scalability, sensitivity, and robustness of our algorithm we performed empirical evaluation on synthetic data. We evaluate the quality of approximation that we can achieve with our greedy algorithm. We also compare our results with the alternative algorithm of removing *all* inconsistent instances.

1) *Synthetic Dataset Generation:* We implemented an uncertain relation generator which lets us generate uncertain relations under pre-specified settings for different parameters. The key parameters are described in Fig 13 below. The generation parameters allow us to control and configure various factors such as the size of the uncertain relation, uncertainty in the uncertain relation, kinds and number of ICs, the "dirtiness" of the relation i.e., the degree of inconsistency in the original relation etc.

The generation of an uncertain relation with constraints comprises of the following basic steps: 1) Generate an ini-

# Tuples	# ICs	Marginals (ms)	IC Resolution (ms)
100	5	< 1	703
1000	50	< 1	4922
10000	500	48	99167 (2 min)
50000	2500	1078	2591384(53 min)

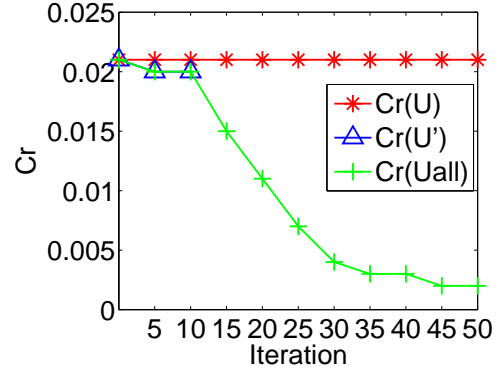
TABLE IV
TIME VS RELATION SIZE

tial (clean) uncertain relation according to the relation size, schema size, and relation uncertainty degree parameters. This includes the definition of a probability distribution over the uncertain relation. 2) Generate specific ICs at attribute, tuple, and relation levels based on the number of ICs parameter. 3) Inject instances of violations for the attribute, tuple, and relation level ICs in randomly chosen attributes, tuples, and sets of tuples (respectively) according to the degree of dirtiness parameters.

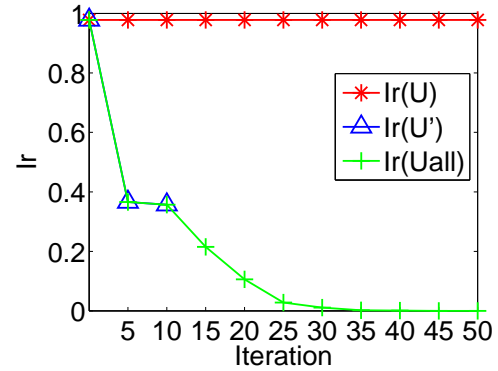
2) *Experimental Results:* On a synthetic dataset of 1000 tuples with 25 ICs (of different types) Figure 14 demonstrates the consistent (Cr) and inconsistent mass (Ir) in the approximation as a function of the number of ICs (iterations) applied. Figure 15 illustrates (for 2 cases of different initial consistency) the approximation quality as a function of the IC iterations applied. We applied the ICs in order that the greedy algorithm selects them, the greedy algorithm terminates according to the utility based criterion whereas the brute force algorithm of resolving all ICs runs on. These results are typical of the many traces we conducted. We clearly see the superiority of our greedy IC selection algorithm (U') which terminates when resolving ICs is no longer beneficial, as opposed to the brute force approach (U_{all}) of resolving all ICs that can cause the quality to significantly degenerate.

In Figure 16 we illustrate the sensitivity of approximation quality (shown averaged over several traces) to (a) the initial consistent mass in the relation, and (b) the degree of uncertainty in the original uncertain relation - which is controlled by the MAX parameter. We observe (a) that uncertain relations of higher original consistency result in better quality approximations, whereas (b) quality depends on other factors such as the degree of inconsistency, constraint distribution etc., as opposed to relation uncertainty defined in terms of the number of attribute value choices MAX .

Figure 17 shows the advantage of using a richer multirow model where we can see that the approximation quality increases as more rows are provided for a multirow representation of each tuple. Finally in Table IV we present the time required for approximation generation with increasing tuples and IC violations, where we show the time for marginals computation and the (total) IC resolution time. Note that once the approximation has been generated we can answer queries very efficiently on the resulting approximation as the constraints have been factored in. The approximation generation times show that our approach is scalable to large datasets. The experiments were conducted on an IBM XSeries.445 machine



(a) Consistent Mass Retained



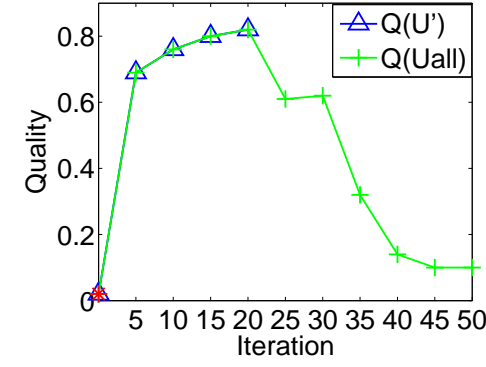
(b) Inconsistent Mass Retained

Fig. 14. Cr and Ir in each iteration

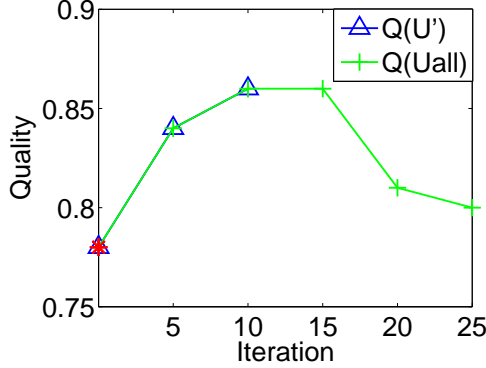
with 4 Intel Xeon 3 GHz processors, 17GB Ram running Windows Server 2003. We must also mention that we have been unable to provide comparative experimental results with a related system such as MayBMS (in particular) as the "assert" operation meant to materialize a database recalibrated given an IC is not provided in the current system.

VIII. RELATED WORK

Probabilistic databases have been an area of activity since the 1980s with foundational works such as [17], [18] extending the relational model and algebra to represent and support uncertainty in databases. Current active projects - MystiQ [9], MayBMS, Trio, or Orion [2] employ different underlying uncertain database representation formalisms that either vary subtly, or in some cases significantly across each other, for instance MystiQ using "or-tuples", Trio using or-sets but with additional "lineage" information, and MayBMS using more expressive world set decompositions (WSDs). MayBMS has considers conditioning probabilistic databases with ICs which is motivated from a data cleaning perspective, dealing with "equality generating dependencies" (equivalent to the tuple level ICs) and just functional dependencies (FDs) from amongst relation level ICs (as opposed to the larger class of relation ICs that we address). Their approach to resolving ICs is quite different from ours. Instead of applying ICs to

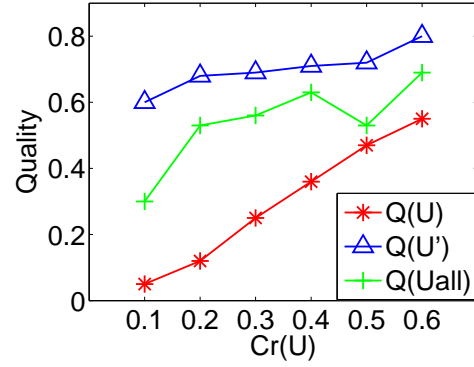


(a) Quality

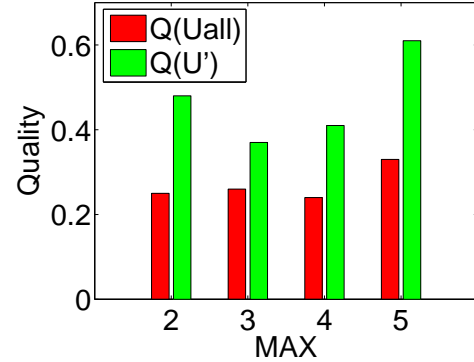


(b) Quality

Fig. 15. Quality after resolving each IC



(a)



(b)

Fig. 16. Sensitivity

an uncertain database as we do, they augment queries with the ICs so that the ICs are resolved at query time. The approach to factor in FDs using a chase based procedure [5] can result in an exponential blow up even with a single FD. Each relation is represented as decomposed into multiple "components" the product of which yields the entire relation. Each component essentially contains the values of an attribute or a set of attributes. Their algorithm is to consider pairs of tuples violating the FD, take each attribute in the FD and merge the components containing those attributes for the pair of tuples into a new component, and then clean the new component by eliminating attribute value combinations that are inconsistent with the FD. In the case of a relation R , with FD $A \rightarrow B$, and pair-wise violations $(t_1, t_2), (t_2, t_3), \dots, (t_{K-1}, t_K)$ with this FD, we will end up with a component that has as columns $(t_1.A, t_2.A, \dots, t_K.A, t_1.B, \dots, t_K.B)$ and in the rows of this component have all consistent combinations of attribute A and B values. The size of this component is $O(M^K)$ where M is the degree of uncertainty (choices) in the attributes. Further, the chase based procedure must select the consistent combinations only and its complexity is also $O(M^K)$. Even with modest values of say $M=2$ and $K=30$, M^K is extremely large. While we observe that their approach is exponential, we note that the authors essentially meant the technique to be used in the context of data that has only *very few*

violations, in which case their approach will work fine. This is substantiated by their experiments which have been done with a degree of data dirtiness as low as 0.001% - 0.005% and also stated as a valid assumption by them given the focus on data cleaning applications. In contrast, our approach is applicable to databases with a much higher degree of data dirtiness, for applications such as information extraction where literally *all* fields in the data can be uncertain i.e., with a degree of dirtiness of 100% ! Also in our synthetic data evaluation we have used an α (dirtiness) factor of at least 5% (Table IV). To the best of our knowledge our work is the first to a) Provide an approach to factoring a large class of ICs, including many kinds of relation level ICs such as FDs, aggregation constraints, inclusion dependencies, and set constraints in a correct manner into an uncertain database, b) Provide an approach to incorporating ICs that makes no assumptions on factors such as the degree of data dirtiness and is thus applicable to applications where the degree of data dirtiness can in practice be quite high.

In information extraction, the approach developed in [1] is to approximate a complex CRF distribution that represents text segmentation possibilities into a probabilistic relational model. This work however does not consider any dependencies *across* different extracted segments, where each extracted segment is treated as a tuple. We address such dependencies

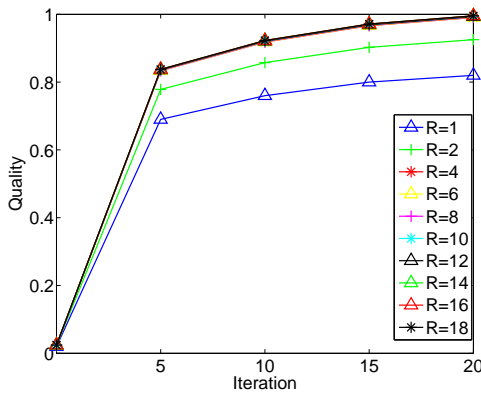


Fig. 17. Quality in Multi-row Model

as relation level ICs. In [1] the probability distribution being approximated is *known* to be generated from a CRF and an efficient forward-backward-message-passing algorithm is employed for marginal computation, vs our setting where marginal probabilities must be estimated. We compared with database repair [10], [19], [20] earlier and further note that most prior work on repair has considered only a limited set of constraints, such as [10] which deals with only functional (FD) and inclusion (IND) dependencies whereas our paper addresses a large class of attribute, tuple, and relation level ICs. Work on consistent query answering (CQA) deals with a related but different problem of answering queries over a dirty database considering constraints over the database - this is established as a hard problem in general [20] with practical approaches [21] provided considering only primary key constraints.

IX. CONCLUSION

We have developed an approach for incorporating integrity constraints into uncertain relations by approximating the uncertain relations. There are several interesting directions for future work, including considering more expressive uncertain database representation models, that we are working on.

REFERENCES

- [1] R. Gupta and S. Sarawagi, "Creating probabilistic databases from information extraction models," in *VLDB*, 2006, pp. 965–976.
- [2] D. Suciu and N. Dalvi, "Foundations of probabilistic answers to queries," in *Tutorial at ACM SIGMOD*, 2005.
- [3] P. Agrawal, O. Benjelloun, A. D. Sarma, C. Hayworth, S. U. Nabar, T. Sugihara, and J. Widom, "Trio: A system for data, uncertainty, and lineage," in *VLDB*, 2006, pp. 1151–1154.
- [4] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*. Addison-Wesley, 1995.
- [5] L. Antova, "Efficient representation and processing of incomplete information," Master's thesis, Saarland University, Feb 2006, <http://www.cs.cornell.edu/~lantova/>.
- [6] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang, "Urank: formulation and efficient evaluation of top-k queries in uncertain databases," in *ACM SIGMOD*, 2007.
- [7] C. Koch and D. Olteanu, "Conditioning probabilistic databases," in *PVLDB 1(1)*, 2008.

- [8] N. N. Dalvi and D. Suciu, "Answering queries from statistics and probabilistic views," in *VLDB*, 2005.
- [9] —, "Management of probabilistic data: foundations and challenges," in *PODS*, 2007, pp. 1–12.
- [10] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi, "A cost-based model and effective heuristic for repairing constraints by value modification," in *ACM SIGMOD*, 2005.
- [11] T. Imielinski, S. Naqvi, and K. Vadaparty, "Incomplete object data model for design and planning applications," in *ACM SIGMOD*, 1991, pp. 288–297.
- [12] N. Ashish, S. Mehrotra, and P. Pirzadeh, "Incorporating integrity constraints in uncertain databases (extended)," <http://www.ics.uci.edu/~ashish/techrep>, Tech. Rep.
- [13] R. Karp, M. Luby, and N. Madras, "Monte-carlo approximation algorithms for enumeration problems," *Journal of Algorithms*, vol. 10, pp. 429–448, 1989.
- [14] S. Khuller, A. Moss, and J. S. Naor, "The budgeted maximum coverage problem," *Information Processing Letters*, vol. 70, no. 1, pp. 39–45, 1999.
- [15] A. Papoulis, *Probability, Random Variables and Stochastic Processes*. McGraw-Hill Companies, 1991.
- [16] "Ties: Trainable information extraction system," <http://tcc.itc.it/research/textec/tools-resources/ties.html>
- [17] D. Barbara, H. Garcia-Molina, and D. Porter, "The management of probabilistic data," *IEEE TKDE*, vol. 4, no. 5, pp. 487–502, 1992.
- [18] R. Cavallo and M. Pittarelli, "The theory of probabilistic databases," in *Proc of VLDB*, 1987.
- [19] A. Lopatenko and L. Bravo, "Efficient approximation algorithms for repairing inconsistent databases," *ICDE*, pp. 216–225, 2007.
- [20] J. Chomicki, "Consistent query answering: Five easy pieces," in *ICDT*, 2007, pp. 1–17.
- [21] A. Fuxman and R. J. Miller, "First-order query rewriting for inconsistent databases," in *ICDT*, 2005.